Program Comprehension with Physical Computing

A Structure, Function, and Behavior Analysis of Think-Alouds with High School Students

Gayithri Jayathirtha
gayithri@upenn.edu
University of Pennsylvania
Philadelphia, PA

Yasmin B. Kafai
kafai@upenn.edu
University of Pennsylvania
Philadelphia, PA

ABSTRACT

Comprehending programs is key to learning programming. Previous studies highlight novices' naive approaches to comprehending the structural, functional, and behavioral aspects of programs. And yet, with the majority of them examining on-screen programming environments, we barely know about program comprehension within physical computing—a common K-12 programming context. In this study, we qualitatively analyzed think-aloud interview videos of 22 high school students individually comprehending a given text-based Arduino program while interacting with its corresponding functional physical artifact to answer two questions: 1) How do novices comprehend the given text-based Arduino program? And, 2) What role does the physical artifact play in program comprehension? We found that novices mostly approached the program bottom-up, initially comprehending structural and later functional aspects, along different granularities. The artifact provided two distinct modes of engagement, active and interactive, that supported the program's structural and functional comprehension. However, behavioral comprehension i.e. understanding program execution leading to the observed outcome was inaccessible to many. Our findings extend program comprehension literature in two ways: (a) it provides one of the very few accounts of high school students' code comprehension in a physical computing context, and, (b) it highlights the mediating role of physical artifacts in program comprehension. Further, they point directions for future pedagogical and tool designs within physical computing to better support students' distributed program comprehension.

CCS CONCEPTS

• **Applied computing → Interactive learning environments**.

KEYWORDS

physical computing, program comprehension, secondary education, electronic textiles

ACM Reference Format:
Gayithri Jayathirtha and Yasmin B. Kafai. 2021. Program Comprehension with Physical Computing: A Structure, Function, and Behavior Analysis of Think-Alouds with High School Students. In *26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2021), June 26-July 1, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3430665.3456371

1 INTRODUCTION

Physical computing is not only increasingly mediating our interaction with the world but is also making a foray into educational spaces. Recently, they have been extensively adopted to introduce computing to K-12 learners across formal and informal settings (e.g., [10, 15]). They provide microcontrollers, sensors, and actuators, and allow learners to design, construct, and program electronic artifacts [2]. In addition, certain Arduino-based tools such as electronic textile construction kits (hereafter e-textiles) integrate traditional practices such as crafting and sewing with computing and support culturally responsive computing while realizing textile-based artifacts [3]. Such an engagement with computation has further diversified participants and products within computing, addressing the disparity in participation across genders, races, and ethnicities within the field to an extent [13, 14].

Program comprehension, the process of developing an understanding of computer programs in a given language, is considered one of the key aspects of computing learning [23, 24]. Understanding the structural, functional, and behavioral aspects of programs is key for learners to not only reason but also to debug and write successful programs (see [25] for a review). The recent collection of a variety of learning activities to foster program comprehension among learners is an evidence for its key role in programming education [12]. However, studies exploring novices' program comprehension mostly examine on-screen programming environments with barely any considering the affordances and constraints of physical computing environments for comprehending programs. These studies draw predominantly from psychological learning theories that view learning as an individual managing long and short term memories (e.g, [24, 25]). This implies that the role of artifacts within programming environments in supporting novices—central to learning with physical computing—is understudied.

Although learner motivation and engagement are extensively studied within physical computing, how these environments support learning programming is unexplored [11]. Knowledge about circuits and fluency with associated representations such as circuit drawings can support program comprehension within physical computing. An understanding of the functionality of different circuit components such as lights, motors, and sensors can provide a context to understand programs. At the same time, these circuits allow for embodied interactions such as pressing buttons or activating sensors and observing light patterns, different from interacting with on-screen program text and artifacts. These will only broaden the different knowledge bases and experiences that learners can employ to comprehend programs.

Distributed intelligence [20] is an analytical framework that accounts for the different elements in a learning environment and allows examining implications of their designs on learning. Such a theoretical framing helps us account for material interactions and squarely fits our focus to examine program comprehension within a physical computing context where meaning-making will be distributed across the learner, programming language elements, and physical artifact that embodies the circuit [20]. Such an investigation will have lessons for pedagogical and tool design for novices. Motivated by this, we conducted think-aloud interviews [8] with 22 high school students individually interacting with a given text-based Arduino program and an accompanying functional e-textile artifact. We qualitatively analyzed videos and annotated transcripts to answer: 1) How do novices comprehend the given program? And, 2) What role does the physical artifact play in the process?

2 BACKGROUND

Prior studies articulate program comprehension as learners attending to three key aspects of programs: structural text, dynamic behavior, and function of programs [5, 25] (see Table 1). Structural aspects include perceivable elements such as indentation, program text and organization, and other syntactic features; behavioral aspects correspond to the abstract program dynamics i.e. the control and data flow or how the programmed system transitions between different states during program execution; and, functional aspects correspond to the purpose or goal of the program. One can attend to these aspects within programs across different granular levels: a.) atomic: individual literals such as constants (e.g., HIGH/LOW, INPUT/OUTPUT in case of Arduino) and statements (e.g., *delay()* or *loop()*); b.) blocks: syntactic or logical groups of statements (e.g., chunks of code enclosed within conditional blocks); c.) relations: connections between groups of statements or code segments (e.g., groups of conditional blocks); and, d.) macro: the overall program.

More recently, Schulte and colleagues [25] extended this model to accommodate yet another key dimension that learners inevitably include while comprehending programs: prior knowledge stemming from programming experiences or the knowledge of the application domain, to name a few. For instance, studies have observed students transferring their learning from one programming environment to another, especially when learning in succession [18]. In addition to programming knowledge, some studies have also highlighted the role of English and mathematical semantics in shaping comprehension (e.g., [7, 22]). From conditional *if* statements to the assignment operation, prior English and mathematical knowledge shape students' understanding of a range of programming elements. Further, in the case of physical computing, domain knowledge of different circuit elements and their functionalities may further contribute to program comprehension (e.g., [6]). Overall, comprehending computer programs demands learners to employ a wide variety of resources while attending to different aspects of programs.

Yet another point to note, most of the above-mentioned studies have examined code comprehension among learners with some familiarity of the programming environment; very few have explored this process among novices within unfamiliar programming environments. Nevertheless, such an understanding is key for a few different reasons: we know from constructivist learning theories that learners are not blank slates and that they come with some prior knowledge that they apply to understand new learning materials. This is evident in Ko and Uttl's [16] study where undergraduate learners' program comprehension in an unfamiliar environment was shaped by their prior programming and problem-solving experiences. Further, a more recent study [29] noted distinct patterns among post-secondary students in how they related the syntactic and semantics aspects while switching programming environments. Studying high school students' preconceptions is important when programming environments are inviting a broad variety of knowledge bases and experiences to action, as in the case of physical computing. However, such studies are rare although they can have valuable insights for teachers and tool designers.

Given how complex program comprehension is, earlier studies have noted significant challenges that novices face while doing so. For instance, Lister and colleagues [19] observed that most novices attended to the atomic aspects: individual literals or statements while struggling to comprehend the program overall. This is similar to novices' bottom-up approach of attending to individual tokens while comprehending programs [25]. The perceivable structural aspects such as punctuation or program organizations are more accessible for novices to grasp, while functional and behavioral aspects might be more challenging due to their abstractness and invisibility [5]. However, all the above-mentioned studies, similar to the field at large, have focused on on-screen programming environments where all the clues are embedded within on-screen program text and environment. With programming contexts significantly shaping opportunities for comprehending programs [25], there is a need to separately explore program comprehension when programming environments include tangible circuits in addition to program text.

Table 1: Schulte's [24] BLOCK Model for program comprehension showing key aspects of programs (structure, behavior, and function).

Duality	***Structure***	***Behavior***	***Function***
Macro	Overall structure of program text	Algorithm of the program	Overall goal of the program in context
Relations	References between blocks	Sequence of method calls or blocks	Relating subgoals to the goal
Blocks	Syntactic or Semantic Regions of Interest	Operation of the Region of Interest	Function of the block, or the sub-goal
Atoms	Language elements	Operation of a statement	Statement Function or Goal

Situated theories of learning help us attend to contextual aspects more closely, providing a framework to study material interactions within physical computing environments. For instance, meaning-making process will be distributed across the given program text, the circuitry embodied in the interactive physical artifact, knowledge about circuits, and other associated representations [20]. Unlike traditional views of cognition and intelligence that places thinking in an individual's mind, distributed perspective brings to the fore environmental aspects to account for "artifactual, physical, symbolic, and social" structures across which intelligence will be distributed [20] (p. 53). From labels on circuit components to programming language tokens, symbols and signs embedded across the space can invite learners to observe and interact with them while understanding programs. On the other hand, a lack of such frameworks may consider program comprehension as a tool-free mental activity and discount the role of physical functional artifacts.

Physical computing construction kits can support programmers comprehend programs in ways different from traditional on-screen programs. A recent study [1] highlighted how middle school students learned Arduino programs in relation to the voltage differences in digital signals. Similarly, Pennington [21] noted that the physical aspects such cables and their connections served as visible domain knowledge for engineers to interpret programs. Physical computing construction kits are comparable to these engineering devices in that the physical artifact reveals the circuitry being programmed. For example, in e-textiles, the different inputs, actuators, the microcontroller, and the connections between them are laid out in the artifact whose outcome is tied to the program [1, 3]. The visibility of the connections between circuit components, prior understanding of their functionalities, and their ability to allow for both observational and interactive engagement can potentially scaffold program comprehension, which has been under-explored. In this study, we took a step towards bridging this gap by examining novices' thinking and interactions as they comprehended programs within a physical computing environment where the program text was accompanied by a functional, physical artifact.

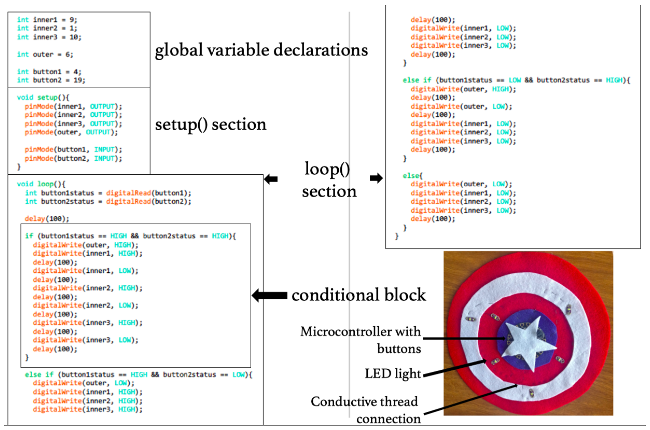
3 METHODOLOGY

3.1 Context and Participants

The study was conducted at a public charter high school located in a large U.S. west coast city. Participants were in the *Exploring Computer Science* class [9], offered to students with no prior formal computer science experience. More specifically, 17 of the 22 interviewees confirmed that they had no programming experience outside of this unit while 5 of them acknowledged some programming experience in after-school clubs or as a part of middle school coursework. As a part of this study, we interviewed 22 consented students (14-18 years; 11 male, 11 female; 8 identified as White, 6 Latino/Hispanic, 6 Asian, and 2 African-American) as identified by the teacher two weeks before an Arduino unit within the curriculum. While all the students had HTML scripting and Scratch programming experiences as a part of the unit, this was their first time interacting with Arduino programs.

3.2 Data Collection and Analysis

The first author conducted and video-recorded think-aloud interviews of each of the 22 students [8] where every student was invited to reason a 56-line Arduino program printed on a paper (the entire program was visible at once) accompanied with an e-textile functional artifact (see Figure 1). The program included traditional Arduino sections: global variable declarations, a setup(), and a loop() with four conditional blocks (each causing a light pattern in response to one of the four two-button press configurations as expressed by logical AND statements). It had meaningful variable names, indentation, and structure, similar to plan-like programs [26] to keep the program accessible to novices. Given that the students were new to text-based Arduino programs, we took care not to overwhelm novices with multiple programs. Interviewees were initially presented with non-powered physical artifact in addition to the printed program, and asked questions such as "how is the code related to the artifact?" and "do any parts of the code stand out?" Then, the artifact was powered with a battery and students were encouraged to interact with the same and think aloud as they reasoned the given program. Unlike the traditional program comprehension interview protocols, We provided a functional artifact along with the code to study the role of artifacts in this process.



```
int inner1 = 9;
int inner2 = 1;
int inner3 = 10;

int outer = 6;

int button1 = 4;
int button2 = 19;

void setup(){
  pinMode(inner1, OUTPUT);
  pinMode(inner2, OUTPUT);
  pinMode(inner3, OUTPUT);
  pinMode(outer, OUTPUT);

  pinMode(button1, INPUT);
  pinMode(button2, INPUT);
}

void loop(){
  int button1status = digitalRead(button1);
  int button2status = digitalRead(button2);

  delay(100);

  if (button1status == HIGH && button2status == HIGH){
    digitalWrite(outer, HIGH);
    digitalWrite(inner1, HIGH);
    delay(100);
    digitalWrite(inner1, LOW);
    delay(100);
    digitalWrite(inner2, HIGH);
    delay(100);
    digitalWrite(inner2, LOW);
    delay(100);
    digitalWrite(inner3, HIGH);
    delay(100);
    digitalWrite(inner3, LOW);
    delay(100);
  }

  else if (button1status == HIGH && button2status == LOW){
    digitalWrite(outer, LOW);
    digitalWrite(inner1, HIGH);
    digitalWrite(inner2, HIGH);
    digitalWrite(inner3, HIGH);
```

```
    delay(100);
    digitalWrite(inner1, LOW);
    digitalWrite(inner2, LOW);
    digitalWrite(inner3, LOW);
    delay(100);
  }

  else if (button1status == LOW && button2status == HIGH){
    digitalWrite(outer, HIGH);
    delay(100);
    digitalWrite(outer, LOW);
    delay(100);
    digitalWrite(inner1, LOW);
    digitalWrite(inner2, LOW);
    digitalWrite(inner3, LOW);
    delay(100);
  }

  else{
    digitalWrite(outer, LOW);
    digitalWrite(inner1, LOW);
    digitalWrite(inner2, LOW);
    digitalWrite(inner3, LOW);
    delay(100);
  }
}
```

global variable declarations
setup() section
loop() section
conditional block
Microcontroller with buttons
LED light
Conductive thread connection

Figure 1: Annotated Interview Artifacts: The Arduino program printed on paper and the physical e-textile project

Guided by the distributed intelligence lens [20], the first author transcribed and annotated each video with student utterances, gestures, and interactions to capture students' meaning-making across the given artifact and the program text. These transcripts and student drawings were qualitatively analyzed in two phases, initial deductive analysis followed by inductive analysis to generate themes [4]. Every program-related instance in each transcript was deductively coded to recognize the structure, function, and behavior, and the level of detail [24]. For instance, students pointing to the visual blocks of code without delving into its role or dynamic behavior was coded as structural, stating the goals of individual blocks as functional, and discussing details by drawing on the data or control flow as behavioral. Further, the nature of interaction with the physical artifact, if any, was noted with each (active seeing or noticing, or interactive engagement). The generated codebook with definitions and examples was shared with another researcher for validation and feedback. A codebook with revised definitions and more examples was applied to 3 of the annotated transcripts (10% of the data) by the first author and yet another researcher independently. Upon reaching close to 90% agreement, disagreements were discussed and the codebook was revised to include clarifications, and the first author coded the rest of the 19 annotated transcripts. This led to a total of 119 episodes of students attending to one of the three aspects of programs across different granularities and modes of engagement with the physical artifact (blue and orange circles in Figure 2). Further, descriptive notes were taken to capture any prior knowledge whenever visible and these were inductively analyzed by the first author independently and included to explain the patterns in the findings appropriately.

4 FINDINGS

We observed patterns similar to traditional program comprehension activities: close to two-thirds of the students (14 out of 22) attended to the structural and functional aspects of the program, and fewer (8 out of 22) extended to reason the program run-time behavior (see Figure 2). Most of them initially focused on the atomic details, either the individual literals or statements, and only later attended to program blocks and the overall program (arrows in Figure 2). They constantly drew from their prior programming experiences and contextual clues from the program and the artifact to further interpret larger segments of the program.

Throughout the process of comprehending the program, the physical artifact afforded two qualitatively distinct modes for engagement: a.) active engagement in the form of observing different components, inferring functionality and tracing the physical connections on the artifact (blue dots in Figure 2), and, b.) interactional engagement involving button presses and relating the light patterns to the program text (orange dots in Figure 2). Different components of the artifact allowed students to employ their prior understanding of circuit elements and to draw from the broad variety of contextual and visual clues. While active engagement supported structural and functional understanding of atomic program components, iterative interactive engagement with the artifact furthered functional understanding at the block, relational, and macro level. Since we found that the program comprehension process was constantly informed by the engagement with the artifact, we will present our findings about program comprehension across structure, function, and behavior while highlighting the role of the physical artifact at each level. Furthermore, we will also share details regarding other resources that students drew from whenever applicable.

4.1 Understanding by Seeing: Structural Comprehension Mediated by Active Engagement

During the initial phase of comprehension, a majority of the students (21 out of 22) attended to perceivable structural aspects, examining atomic program literals. Very few (3 of 22) commented on either the relational or macro levels (the Structure column in Figure 2). An active engagement with the artifact while looking, observing, and tracing physical connections provided a particular

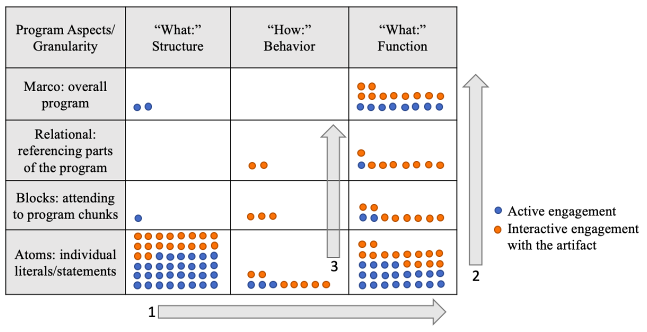


Figure 2: The visualization of program comprehension episodes across program aspects and granularity while actively (blue) and interactively (orange) engaging with the artifact; the arrows indicate the order of students' attention.

context for students to understand certain literals. Students further drew on a variety of other visual clues and employed their prior programming experiences to explain certain other program structures. However, structures unrelated to circuit elements or to students' prior experiences were left unattended.

The visibility of different components and the interconnections between them in the physical artifact allowed students to map keywords and variable names to corresponding artifact elements. Based on their observations of different circuit components, students connected variable names such as button1, button2, button1Status, and button2Status onto the corresponding perceivable elements i.e. buttons on the artifact. Further, some students (9 of 22) traced the connections between the microcontroller and the lights, and concluded that the microcontroller was sending HIGH and LOW signals to affect brightness of lights, pointing to the HIGH/LOW in the program. A few other students associated the buttons and lights to INPUT and OUTPUT keywords in the program. Further, the spatial positioning of lights along the inner and the outer rims of the shield artifact led some students (6 of 22) to relate the variables *inner* and *outer* to them. However, not all visual clues led novices to better understand the program. For instance, some (3 of 22) naively thought of the six lights as connected to the six variables in the program and a few others affiliated HIGH and LOW to the positioning of lights along the higher and lower rims of the circular artifact (see Figure 1, right). Nevertheless, a constant quest for visual clues to map different aspects of the physical artifact with the perceivable program elements was a common first step towards comprehending the program.

Furthermore, program elements such as braces and indentation visually structured the program as six chunks (see Figure 1, left). Some students (5 out of 22) recruited their prior Scratch programming experience and made a one-to-one correspondence between program chunks and individual lights in the artifact—similar to how *sprites* relate to code blocks in Scratch. For instance, Chad (a student) thought of each program chunk as located "inside of the [light]," pointing to them in the artifact. Similarly, Gala explained that the "little lights" would "read" these chunks to understand if they should "turn on, or how bright they should be." The very formatting of text led a few students (4 out of 22) to relate the given program to HTML scripts, yet another programming experience that they had. For instance, Gala recollected HTML as writing scripts so that "if you click your mouse over something, [the computer] would turn a different color or [the computer] would take you to a different page" and saw that the given program "kinda looked like [that]." But, a significant pattern was an overall lack of attention to global variables or the *setup()* module at the beginning of the program. Students selectively attended to certain structural aspects, mostly limited to familiar and perceivable literals or chunks of code, while disregarding other code structures which did not directly map onto the circuit elements, or their prior experiences and knowledge.

4.2 Understanding by Doing: Functional Understanding Mediated by Interactive Engagement

Unlike the structural understanding of the program, most of the functional understanding was mediated by interacting with the physical artifact. A majority of the students (17 out of 22; rightmost column in Figure 2) comprehended program functionality across different granular levels—from individual statements to the overall program—by not only observing the given artifact but also closely interacting with it. While observing artifacts afforded students opportunities to comprehend the functionality at an atomic level, interacting with the artifact further allowed comprehending specific program blocks, the relationship between blocks, and their articulation of the overall goal of the program.

Paying close attention to the different circuit components on the artifact allowed students to predict the functionality of certain individual program statements. Noticing buttons and lights in the artifact supported students' meaning-making of tokens such as INPUT/OUTPUT and HIGH/LOW, and literals such as *buttonStatus* within the program. Further, close to half of the students not only recognized buttons as inputs and lights as outputs, they connected the two possible states of these devices (on or off) to the program literals HIGH and LOW. This helped half of the students (11 of 22) further explain the functionality of certain statements. For example, Gala understood HIGH and LOW constructs as related to the brightness of lights and applied this to extend her explanation initially to *digitalWrite* statements and later to a conditional block. "Since they are all LOW, that means the lights are not on," she said pointing to the *digitalWrite* statements with LOW as parameters in the last conditional block. This further expanded her understanding to explain different conditional statements, evaluating logical expressions in terms of the statuses of the two buttons.

In addition to observing the physical artifact, closely interacting with it further provided opportunities for novices to understand the functionality of certain blocks. One obvious and common example was how students extended their functional understanding of conditional statements as they pressed buttons and observed particular light patterns. For instance, Adam explained the conditional statement as "checking" the buttons for particular positions (on or off) and accordingly causing light patterns on the artifact. He derived this explanation based on his repeated interaction with the buttons (see Figure 3), pressing to cause different configurations while observing the changing light patterns on the artifact. Upon explaining the individual functionality of each of the four conditional blocks, Adam even described the variable declarations and the *setup()* sections as "generic," implying they are not related with the button presses unlike the rest of the program. In sum, iterative interaction with the artifact furthered functional comprehension to explore beyond the atomic level details of the program.

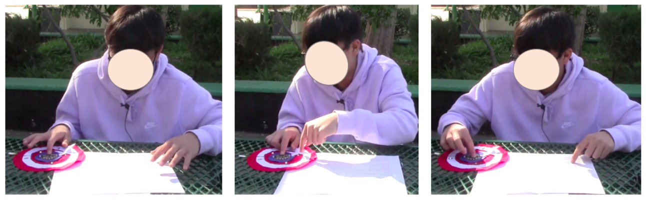


Figure 3: Adam interacting the buttons, observing the light patterns, and connecting them to explain the functional aspect of conditional blocks of the program.

4.3 The Invisibility Issue: The Limited Role of the Artifact in Supporting Behavioral Understanding

Unlike the structural and functional program aspects that were mediated by the artifact, the invisibility of the underlying dynamics led to only a small proportion (8 out of 22) of the students pointing at program run-time behavior. Most of their explanations were limited to individual statements while a few (3 out of 8) extended them to describe operations of program blocks and relationships between them (see the Behavior column in Figure 2). For the most part, the abstract run-time program behavior i.e. comprehending data and control flow was inaccessible to the novices.

Among students who went far enough to reason behavior of certain statements, they did so by drawing on their prior knowledge from English and Scratch programming language semantics. Even here, a few statements and tokens were more accessible to novices than others. For instance, Bash articulated the behavior of *digitalWrite()* statements as the computer "digitally writing things" to the lights while others assumed it as sending directions to the lights. Similarly, Chad translated the *loop()* statement as "constantly checking the buttons" while others assumed similar behaviors without explicit mentions. Such support from the English language semantics was also evident in how four of the seven students explained *if-else* statements in controlling the execution flow of the program as similar to the "if " and "else" in the English language. Further, students explained *delay()* as causing "the lights to wait" to make different patterns, attributing their explanation to the behavior of the wait block in Scratch programs.

However, extended interaction with the functional artifact led a few students (3 out of 8) go beyond individual statements to reason chunks or blocks of code. Nyla was one such rare case, where she constantly interacted with the artifact to understand the given program beyond functional descriptions to trace program execution. Similar to Adam, she repeatedly interacted with the artifact but paying attention to the specific changes in the data and control flow. She said she wanted to "figure out which one is *button1*, which one is *button2*; and, which inner [variable] corresponds to which light," clearly recognizing the mapping between different parts of the code and the circuit. She further explained the working of different conditional blocks in the program, speaking from the perspective of the computer and establishing how the computer understands HIGH and LOW in the program based on the statement in which it is used. Nyla recognized that HIGH and LOW in the case of buttons implied being pressed or not while in the case of lights implied turning on and off respectively. These combined with her understanding of the *if* statement (as drawn from Scratch blocks) got her to explain the behavior of one of the conditional blocks, and of *else-if* statements as mutually exclusive blocks of programs that will be executed depending on the "status of the buttons." While the contextual information in addition to interacting with the artifact supported a few students' understanding of program's behavior, it was far fewer compared to many more students who generated functional or structural explanations, pointing to the invisible nature of these ideas and a need to intentionally integrate them in pedagogy.

5 DISCUSSION

Novices' program comprehension was clearly distributed across materials, space, and time in the form of program text, the physical circuitry, and their prior programming and natural language experiences. Their limited success with reasoning program behavior calls for explicit integration of *notional machines* within programming education [27], while the distributed nature of program comprehension invites careful design of tools, environments, and programming languages for novices.

5.1 Program Behavior in Programming Pedagogy

Novices demonstrated an ability to comprehend the structural and the functional aspects of the program by drawing clues and supports from a variety of resources. And yet, similar to observations in prior studies [5, 6, 25], the invisible abstract program behavior was hard to comprehend. This points to a need to explicitly support novices in computing classes to understand run-time program behavior. Guiding students to understand program dynamics at appropriate levels of abstractions, also called notional machines [7], helps develop a robust understanding of program execution and build their capacity to comprehend, debug, and write programs successfully. This matches with Sorva's [27] recent call for programming pedagogy to integrate notional machines to specifically support novices to understand the abstract program behavior. Findings above allude to a variety of probable notional machines based on the different resources that students recruited while reasoning the program, drawing analogies from prior programming experiences and contextual details, to mention a few. Such a design and analysis will also be one of the first accounts of the nature and role of notional machines within physical computing [17].

5.2 Distributed Intelligence in Tool and Language Design

The distributed nature of program comprehension points to certain directions for future tool design. While observing the artifact allowed for structural mappings between programming language elements and the context, interacting with a artifact allowed students to infer functional meanings for different parts of the program. Though learners do not generate a functional artifact at the beginning of their learning trajectory, presenting some representation of artifacts along with program text early on has the potential to support program comprehension. Adopting Pea's [20] framework and viewing program comprehension as a distributed activity can refocus tool design to embed scaffolds and "intelligence" in tools and environments for novices. Programming environments and tool designers can provide interactive representations of functional artifacts that map programming constructs to observable outcomes and support distributed meaning-making within physical computing.

Furthermore, the variety of meanings that students activated, drawn from contextual to prior programming and natural language experiences, illuminate the active role of programming languages in shaping novices' meaning-making. This was visible in the way certain tokens such as *delay()*, *loop()*, and even meaningful variable names rendered to novice program comprehension. Programming language elements that had close mappings to the interactive artifact, such as INPUT/OUTPUT corresponding to buttons and lights, HIGH/LOW to light brightness, and *delay()* to observed light patterns, were more accessible to novices. Further, constructs such as *if-else* conditional statements invited semantics from prior programming and natural language experiences to comprehend the given program. On the other hand, tokens such as *pinMode()* and *int*, were less accessible. All these can be further considered while designing languages for physical computing for novices.

6 CONCLUSION

Although this analysis revealed certain patterns in high school students' program comprehension within physical computing, it was limited in a few ways. Students were provided an opportunity to comprehend only one program, restricting the programming language constructs that could be examined and also binding the context to this particular artifact design with a very specific circuit layout. Further, although some may argue that 22 students is a good enough sample size for qualitative analysis to have overlapping and saturated themes [28], future studies may repeat this process for larger sample sizes and with a variety of programs to verify results.

With physical computing becoming more prevalent in K-12 introductory computing classrooms, this study indicates how these programming environments support novices' code comprehension. While the above-reported findings reaffirm observations in the field, it adds another dimension by highlighting the role of physical artifacts in providing a context and in mediating program comprehension among novices. And, it illuminated its limitation in affording opportunities to reason program behavior, calling for intentional pedagogical and tool design to make transparent the invisible program execution dynamics for novices.

ACKNOWLEDGMENTS

We would like to thank Luis Morales-Navarro and Katie Cunningham for help with data analysis, anonymous reviewers for their feedback, and participating teacher and students. This project was supported by National Science Foundation (#1742140).

REFERENCES

[1] Doug Ball and Colby Tofel-Grehl. 2020. Potentially Electric: An E-Textiles Project as a Model for Teaching Electric Potential. *The Physics Teacher* 58, 1 (2020), 48–51.

[2] Paulo Blikstein. 2013. Gears of our childhood: constructionist toolkits, robotics, and physical computing, past and future. In *Proceedings of the 12th international conference on interaction design and children*. 173–182.

[3] Leah Buechley, Kylie Peppler, Michael Eisenberg, and Yasmin Kafai. 2013. *Textile Messages: Dispatches from the World of E-Textiles and Education. New Literacies and Digital Epistemologies. Volume 62.* ERIC.

[4] John W Creswell and Cheryl N Poth. 2016. *Qualitative inquiry and research design: Choosing among five approaches.* Sage publications.

[5] Kathryn Cunningham, Mark Guzdial, and Barbara Ericson. 2020. Using the Structure Behavior Function framework to understand learning of computer programming. (2020). (Manuscript submitted for publication).

[6] Kayla DesPortes and Betsy DiSalvo. 2019. Trials and Tribulations of Novices Working with the Arduino. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. 219–227.

[7] Benedict du Boulay, Tim O'Shea, and John Monk. 1981. The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies* 14, 3 (1981), 237–249.

[8] Anders Ericsson and Herbert A Simon. 1998. How to study thinking in everyday life: Contrasting think-aloud protocols with descriptions and explanations of thinking. *Mind, Culture, and Activity* 5, 3 (1998), 178–186.

[9] Joanna Goode, Gail Chapman, and Jane Margolis. 2012. Beyond curriculum: the exploring computer science program. *ACM Inroads* 3, 2 (2012), 47–53.

[10] Steve Hodges, Sue Sentance, Joe Finney, and Thomas Ball. 2020. Physical computing: A key element of modern computer science education. *Computer* 53, 4 (2020), 20–30.

[11] Michael Horn and Marina Bers. 2019. Tangible computing. *The Cambridge handbook of computing education research* 1 (2019), 663–678.

[12] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, et al. 2019. Fostering Program Comprehension in Novice Programmers-Learning Activities and Learning Trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. 27–52.

[13] Gayithri Jayathirtha and Yasmin B. Kafai. 2019. Electronic textiles in computer science education: a synthesis of efforts to broaden participation, increase interest, and deepen learning. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 713–719.

[14] Yasmin Kafai, Kristin Searle, Crîstobal Martinez, and Bryan Brayboy. 2014. Ethnocomputing with electronic textiles: Culturally responsive open design to broaden participation in computing in American Indian youth and communities. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 241–246.

[15] Yasmin B Kafai, Deborah A Fields, Debora A Lui, Justice T Walker, Mia S Shaw, Gayithri Jayathirtha, Tomoko M Nakajima, Joanna Goode, and Michael T Giang. 2019. Stitching the Loop with Electronic Textiles: Promoting Equity in High School Students' Competencies and Perceptions of Computer Science. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1176–1182.

[16] Andrew Jensen Ko and Bob Uttl. 2003. Individual differences in program comprehension strategies in unfamiliar programming systems. In *11th IEEE International Workshop on Program Comprehension, 2003.* IEEE, 175–184.

[17] Shriram Krishnamurthi and Kathi Fisler. 2019. Programming paradigms and beyond. *The Cambridge Handbook of Computing Education Research* 37 (2019).

[18] Colleen M Lewis. 2010. How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 346–350.

[19] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L Whalley, and Christine Prasad. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin* 38, 3 (2006), 118–122.

[20] Roy D Pea. 1993. Practices of distributed intelligence and designs for education. *Distributed cognitions: Psychological and educational considerations* 11 (1993), 47–87.

[21] Nancy Pennington. 1987. Comprehension strategies in programming. In *Empirical studies of programmers: second workshop*. Ablex Publishing Corp., 100–113.

[22] Yizhou Qian and James D Lehman. 2016. Correlates of Success in Introductory Programming: A Study with Middle School Students. *Journal of Education and Learning* 5, 2 (2016), 73–83.

[23] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer science education* 13, 2 (2003), 137–172.

[24] Carsten Schulte. 2008. Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth international Workshop on Computing Education Research*. 149–160.

[25] Carsten Schulte, Tony Clear, Ahmad Taherkhani, Teresa Busjahn, and James H Paterson. 2010. An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports*. 65–86.

[26] Elliot Soloway and Kate Ehrlich. 1984. Empirical studies of programming knowledge. *IEEE Transactions on software engineering* 5 (1984), 595–609.

[27] Juha Sorva. 2020. Chapter 14: Naive Conceptions of Novice Programmers. In *Computer Science in K-12: A A to Z handbook on teaching programming*, Suchi Grover (Ed.). Edfinity, Palo Alto, CA, USA, 143–157.

[28] Keith Trigwell. 2006. Phenomenography: An approach to research into geography education. *Journal of geography in higher education* 30, 2 (2006), 367–372.

[29] Ethel Tshukudu and Quintin Cutts. 2020. Understanding Conceptual Transfer for Students Learning New Programming Languages. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. 227–237.